U.S. Patent Application

Title: **LEADING ZERO ANTICIPATORY (LZA) ALGORITHM AND LOGIC FOR HIGH SPEED ARITHMETIC UNITS**

Inventor: **Yatin Hoskote**

PREPARED BY:

KENYON & KENYON

333 W. SAN CARLOS ST., SUITE 600
SAN JOSE, CALIFORNIA 95110

408-975-7500

2207/11238
Exp. Mail No. EL566656189US

LEADING ZERO ANTICIPATORY (LZA) ALGORITHM AND LOGIC FOR HIGH SPEED ARITHMETIC UNITS

Background of the Invention

[001] The field of the present invention is the field of Leading Zero Anticipators (LZA) that predict the number of leading zeros or leading ones in a sum of mantissas generated by an adding device used to add two floating point numbers.

[002] Leading Zero Anticipators ("LZAs") are typically used to predict the number of leading zeros in a sum of mantissas during floating point addition. Prediction of the number of leading zeroes is used to normalize the result of the addition and is performed at the same time as (in parallel with) mantissa addition, which increases the speed of the normalization process that is performed after mantissa addition has been completed.

[003] In a traditional LZA design shown in Figure 1, mantissaA and mantissaB (labeled as 10) both enter the adder 13 and the LZA 12 simultaneously. A normalization shift is performed by the shifter 15 after addition is performed. LZAs 12 are commonly used in devices performing fast addition that try to reduce the number and length of clock cycles required to do the addition and in devices that implement a single instruction which performs multiplication followed by addition. The use of redundant formats such as the carry-save format during addition does not require the use of the LZA during the addition. But the use of redundant formats does require placement of an LZA in parallel with a second adder which adds the sum and carry bit streams generated by the carry-save addition process. The sum and carry bit streams are added before the final normalization stage is completed. In such circuits, the carry and sum become the input operands (mantissaA and mantissaB) that are transmitted to the adder in Figure 1. The nature of addition in redundant formats invalidates the use of several LZAs in the final add operation. For instance, in the carry-save format, the carry term is

shifted left by 1 bit position (equivalent to multiplying the carry value by 2) before addition is performed. If there is a negative carry term and a positive sum term, the left shift that is performed after carry save addition might shift out the leading one of the carry term, making it a positive number (the leading bit indicates sign). The addition of two seemingly positive numbers (sum and carry) may then produce a valid negative number. This seemingly contradictory result is valid in carry-save addition but invalidates the assumptions made in the designs of many LZAs. For example, consider a carry (value -6) and a sum (value 7) with expected final result –6x2+7= -5. The corresponding 4 bit binary vectors in 2's complement are carry 1010 and sum 0111. When the carry is shifted left 1 bit, it becomes 0100. Adding 0100 (which is now seemingly a positive number) to 0111 (also a positive number) gives 1011 which is a negative number (-5) and is the correct expected final result.

[004] Another drawback of conventional LZAs occurs if the operation is a subtraction. Traditional application of LZAs during subtraction requires that the mantissa of the smaller operand must be subtracted from the mantissa of the larger operand so that a positive result is generated. To insure that the operand of smaller magnitude is subtracted from the greater one, a comparator is used to determine which mantissa is larger. This is expensive because of the need to use extra logic gates. Another traditional method of leading zero anticipation does not require comparison but uses 2 LZAs together which are implemented so that they make opposite assumptions about the sign of the generated result when anticipating the number of leading zeroes or ones. The correct LZA output is then chosen depending on the sign of the result of the addition. This is also expensive.

[005] Yet another conventional implementation of LZAs does not require comparison of mantissas and uses just one LZA. However logic designers implementing this form of LZA must assume that the "normal" rules of addition apply when the operands are added. Consequently, this implementation cannot be used when the input operands are expressed in redundant formats, such as

the carry save format. This is the case because addition in redundant format does not follow all the conventions of normal addition as explained above.

[006] Therefore, what is required is a novel LZA that may be implemented with a single LZA and that is implemented in a way that does not make assumptions regarding rules of addition that preclude the use of redundant formats such as the carry save format. Also, an LZA that does not require comparison of the magnitude of the input operands or knowledge of the sign of the result in order to anticipate the number of leading zeroes or leading ones is needed.

Brief Description of the Drawings

[007] Fig. 1 illustrates an adder circuit using a Leading Zero Anticipator (LZA) in the prior art.

[008] Fig. 2 illustrates an adder circuit and using the Leading Zero Anticipator (LZA) according to an embodiment of the present invention.

[009] Fig. 3 illustrates logic circuitry that implements the LZA according to an embodiment of the present invention.

Detailed Description

[010] The present invention pertains to a method to anticipate the number of leading zeros or leading ones in the result of the addition of two floating point numbers. The present invention may be used to remove leading zeroes or ones from sums produced in arithmetic units. Typically, the present invention will be used to remove leading zeroes or ones from the sum of mantissas of two floating point numbers.

[011] The algorithm and combinational logic of the present invention do not necessarily require a comparison of input operands nor do they need two separate counters in order to count the number

of leading zeros and leading ones as required in most other LZAs. The combinational logic of the LZA of the present invention does not need to incorporate specific assumptions about properties of addition, and may be used in implementations using redundant formats such as the carry-save format that do not follow the rules of conventional addition. The combinational logic presented here can be more efficient than other designs used to perform leading zero and leading one anticipation in that anticipation can be performed more quickly and with fewer components, saving costs.

[012] One possible application of the present invention is shown in Figure 2. This implementation includes a dual adder 23 that performs addition of two mantissas and produces the sum as well as its 2's complement (negative of the sum). If the sum is a negative number, its 2's complement will be positive and is chosen as the final result. An LZA 22 typically includes combinational logic which predicts the position of the first logical 1 (0 for negative result mantissas) and a counter that typically may be one of the last stages of the logic circuitry of the LZA 22. This counter counts the number of leading zeroes (ones for negative result mantissas) in the sum by counting the number of bits to the left of an indicating one in a predictive bit stream. One or more of the bits of mantissaA and mantissaB (20) which are in 2's complement format are transmitted to the dual adder 23 and the LZA 22 simultaneously. Typically, the present invention includes a logic design in which the LZA 22 receives sets of three bits of each input operand simultaneously and generates a vector L (predictive bit stream) that has a leading one in the bit position before the leading 0 or 1 of the mantissa sum ("before" in this context typically means "to the left of"). Any fast counter that counts the number of zeros "before" the leading 1 in the generated vector L may be used as part of the LZA 22. The predictive count produced by the counter is the output data transmitted by the LZA 22 and is used to perform normalization shifting. Normalization shifting is performed by the shifter 25 and is used to remove leading zeroes in a mantissa sum. The same predictive count may be used regardless of

whether the resulting mantissa is positive or negative, i.e. regardless of whether the multiplexer 24 output in Figure 2 is the addition result or its 2's complement.

[013] Figure 2 shows a configuration including the LZA 22 of the present invention. MantissaA and mantissaB are transmitted to the dual adder 23 and the LZA 22 of the present invention. The result of mantissaA plus mantissaB is transmitted to the multiplexer 24 when a dual adder is used. The positive result is chosen (based on the sign of the first sum) and then transferred to the bit shifter 25. Bit shifter 25 shifts the sum to the left by the "count" quantity (the number of leading zeroes in L before the predictive one) calculated by the LZA 22. The "count" quantity indicates a value one less than the number of leading zeroes before the first logical one in the final sum. Shifter 26 is required to do an additional shift to account for remaining one or two leading zeroes.

[014] In a single adder implementation, the mantissas are added, and a positive or negative sum is produced. If the single adder produces a negative result (the sign bit will indicate the sign of the result), then leading ones may need to be shifted out of the sum. If the output transmitted from the adder is negative, another circuit converts it into a positive sum after the shifting. As before, a further shift of the positive value by one or two bits may be required.

[015] The combinational logic of the LZA 22 attempts to predict a change in the bit pattern of the sum of the mantissas such as the following: 1..11→ 0 (sequence of ones followed by a zero) or 0..00→ 1 (sequence of zeroes followed by a one). Examining three bits at a time of each of the input mantissas (starting with the most significant bit) is sufficient to determine if such a transition 0→1 or 1→0 will occur between the first and second bits of the 3 bit group of bits in the sum mantissa that is currently being generated.

[016] A portion of combinational logic circuitry implementing the LZA 22 of the present invention is shown in Fig. 3. Bits from the two mantissas mA and mB labeled 30 are received by the

combinational logic elements 31-36 shown in figure 3. Mantissa bits are received and processed by a variety of logic gates in the LZA. Some of the logic gates/logic elements that receive and process mantissa bits are labeled as follows: the xnor ("X" = A XNOR B) gate 31, the "propagate" ("P" = A XOR B) gate 32, the "generate" ("G" = A AND B) gate 33, the nand ("N" = A NAND B) gate 34, the or ("O" = A OR B) gate 35, and the "zero" ("Z" = NOT A AND NOT B) gate 36 transmit logical output bits for each bit pair of the input operands.

[017] The vector L (37) is generated by these logic gates typically by transmitting the signals mA and mB to the logic gates 3 bits at a time. The most significant 3 bits of each mantissa mA and mB are transmitted to the logic gates pictured in Fig. 3, then the second, third, and fourth most significant bits of each mantissa are transmitted to the logic gates, and so forth. N-2 groups of 3 bits are transmitted to the logic gates, where N is equal to the total number of bits in each mantissa. This process may be repeated until every bit of each mantissa has been transmitted to the circuit shown in figure 3.

[018] When gate 37 or any bit position of vector L transmits a one, a transition 0→1 or 1→0 in the mantissa sum is indicated. The P, G, Z, X, N and O signals are generated for each of the bit pairs of the input operands. The outputs of each logic element in fig. 3 are thus labeled as X or Z or P and so on. For each group of 3 bits, the signals that have value one are used to build a 3 letter pattern. Thus, each of the letters of each pattern correspond to a logic gate labeled with that letter that transmits a high signal. The first letter of the three letter pattern represents a gate in the first stage of Fig. 3, the second letter represents a gate in the second stage, and so forth. Gate 37 generates a one for certain patterns of the 3 bit pairs of the mantissa inputs. For example, pattern ZZG causes gate 37 to generate a one because it reflects a 0 → 1 transition. ZZG represents the addition of the values 001 and 001 with a potential carry bit, C. In adding these two values together, the result would be 01C' meaning

there is a 0 → 1 transition between the first and second bits. The pattern ZGG, on the other hand, causes gate 37 to generate a zero because it does not indicate such a transition. ZGG represents the addition of the values 011 and 011 with a potential carry bit C. In adding these two values together, the result is 11C meaning that there is no 0 → 1 transition between the first and second bits.

[019] With the values Z, P, and G, there are twenty-seven possible patterns for three bit values. Of these values, the following eighteen patterns imply a transition from a leading zero or one when they occur: PGP, PGG, PZP, PZZ, ZZP, ZZG, ZPZ, ZPP, ZPG, GZP, GZG, GPZ, GPP, GPG, GGZ, GGP, ZGZ, and ZGP. The "P", "G", "X", "Z", "N" and "O" signals may be tapped from the adder 23 shown in Fig. 2 if the proper adder is used. The number of logic levels required may also be reduced through the use of complex gates. The vector L that is generated by the LZA 22 may be stored in register 38 and is transmitted to a standard leading zero counter 39 to produce the LZA count. The counter in the LZA 22 may be of any form, but it should be able to count from zero up to the width of the input operands. The least significant bit of the vector L (37) is designated "1" to satisfy the case when the leading zero or one is in the last two bits of the result and therefore cannot be predicted by the LZA 22 logic circuitry.

[020] The vector L has a "1" at the bit position of the predicted transition. The LZA count value is used to shift the addition result left and remove the leading zeroes or ones. The initial prediction of LZA count could be inaccurate by 1 bit position to the left in some cases (specifically the following six of the eighteen cases mentioned earlier: PGP, PZP, ZZP, GZP, GGP, and ZGP). During the addition of the input operands, the carry propagates from the least significant bit (rightmost) to the most significant bit (leftmost). The value of this carry could further increase the number of leading zeroes by one. On the other hand, when the final addition result is negative (leading ones) the 2's complement value out of the dual adder is chosen. This can decrease the number of leading zeros by

one. For example, PGP represents the addition of two values (i.e., 111 and 010, or 011 and 110) with a potential carry bit, C. The result is 0CC'. Thus, if C is 0, then the result is 001 and if C is 1, then the result is 010.

[021] It may be necessary to have a final left correction in the LZA count value by 0 or 1 or 2. This correction is equal to the number of leading zeroes remaining in the final positive mantissa after normalization shifting. An additional shift is then done to remove them. While Fig. 2 and Fig. 3 show a particular embodiment of the present invention, the LZA of the present invention may be used in a variety of applications and contexts.

[022] While certain embodiments of the present invention have been described herein, the present invention should not be construed as being restricted to those embodiments. All embodiments and implementations covered by the claims as amended will be embraced by the present invention.